1998-06

Software metrics: a case analysis of the U.S. Army Bradley Fighting Vehicle A3 Program

Romero, James S.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SOFTWARE METRICS:
A CASE ANALYSIS OF THE U.S. ARMY
BRADLEY FIGHTING VEHICLE A3 PROGRAM

by

James S. Romero

June 1998

Principal Advisor: David F. Matthews
Associate Advisor: Mark E. Nissen

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1998	Master's Thesis

4. TITLE AND SUBTITLE SOFTWARE METRICS: A CASE ANALYSIS OF THE U.S. ARMY BRADLEY FIGHTING VEHICLE A3 PROGRAM

5. FUNDING NUMBERS

6. AUTHOR(S)
Romero, James S.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Software development efforts have become the highest-risk element of modern program management. One way that we can mitigate this risk is through the use of metrics. Software metrics can give us insight about the progress, quality, and expected completion of a software development effort. In earlier software development efforts, programming was viewed as a "black art" and, consequently, software metrics were not commonly used. Today, it is generally accepted that a software development effort should be properly planned and that software metrics should be used to control the project. Program managers are no longer concerned about whether or not to use metrics, but are more concerned with which metrics to use and whether or not the ones chosen will be effective. The Bradley Fighting Vehicle A3 Program provides valuable insight into the use of metrics. A principal finding of this research is that implementing an effective metrics program is extremely difficult, especially when the contractor is not experienced in developing software-intensive systems. Because this situation often exists, future and current program managers must assess their own knowledge of software development and plan to mitigate the effects of other factors they cannot influence. They must educate themselves on software issues and metrics and solicit assistance from independent agencies that specialize in software development.

14. SUBJECT TERMS
Software Development, Software Metrics, Bradley Fighting Vehicle A3, MICOM Software Engineering Directorate (SED)

15. NUMBER OF PAGES
80

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

SOFTWARE METRICS: A CASE ANALYSIS OF THE U.S. ARMY BRADLEY FIGHTING VEHICLE A3 PROGRAM

James S. Romero
Captain, United States Army
B.S., United States Military Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
June 1998

ABSTRACT

Software development efforts have become the highest-risk element of modern program management. One way that we can mitigate this risk is through the use of metrics. Software metrics can give us insight about the progress, quality, and expected completion of a software development effort. In earlier software development efforts, programming was viewed as a "black art" and, consequently, software metrics were not commonly used. Today, it is generally accepted that a software development effort should be properly planned and that software metrics should be used to control the project. Program managers are no longer concerned about whether or not to use metrics, but are more concerned with which metrics to use and whether or not the ones chosen will be effective. The Bradley Fighting Vehicle A3 Program provides valuable insight into the use of metrics. A principal finding of this research is that implementing an effective metrics program is extremely difficult, especially when the contractor is not experienced in developing software-intensive systems. Because this situation often exists, future and current program managers must assess their own knowledge of software development and plan to mitigate the effects of other factors they cannot influence. They must educate themselves on software issues and metrics and solicit assistance from independent agencies that specialize in software development.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to study software metrics from the program manager's perspective. To accomplish this I focus on the application of metrics in the software development effort for the M2A3 variant of the Bradley Fighting Vehicle. Specifically, I analyze the effectiveness of software metrics in this program and the reasons for their effectiveness. This analysis also illustrates lessons learned during this software development with respect to metrics. The objective of this thesis is to discuss and generalize from these lessons learned, thereby informing and benefiting future program managers of software-intensive systems.

B. BACKGROUND

One of the primary strengths of the United States Armed Forces is advanced technology. This was never illustrated more clearly than when our forces dominated the battlefield during the Gulf War. There are many varied technologies that contribute to the effectiveness of our weapon systems; however, software can be viewed as a common thread. Today, our fighter aircraft would not fly and our venerable M1 Tank and M2 Bradley Fighting Vehicle would not "hit the side of a barn" without software. As we continue to modernize our force we are increasing our use of software to include systems which were traditionally viewed as being too simple to need it. An example of this is the Land Warrior Project where a fully-integrated combat system is being developed for the fighting soldier. This software-intensive system will include a heads-up monocular

display which will incorporate navigation including graphical maps and global positioning system, communications, individual weapon fire control, and night vision capabilities.

As we increase use of software in the Department of Defense (DoD) we are also increasing our expectations concerning what software can accomplish. We expect software to increase the capabilities of our weapon systems, to make them more flexible, and to allow us to upgrade their capabilities in the future. This increase in expectations has caused the size and complexity of our software to increase dramatically. As size and complexity increase, the challenges associated with software program management grow exponentially. In fact, poor management is cited as the primary cause of failures in software-intensive systems [Ref. 32, Ch. 1, p. 18]. While expectations increase and program management becomes more difficult, software technology continues to evolve. New software languages, which support tools and development methods, are constantly changing, making previously valid software practices and experience obsolete. For example, we are using 4^{th} and 5^{th} generation languages that are very powerful compared to earlier generation languages. One line of code in a 4^{th} generation language can generate hundreds of lines of code in machine or assembly language, and mixing early-generation re-used code with modern languages and methods is difficult [Ref. 31].

The growing size, complexity, and managerial challenge associated with software, along with DoD's increased expectations, have made software development efforts the highest-risk element of modern program management. One way that we can mitigate this risk and better manage software development efforts is through the use of metrics. In

fact, it would be very difficult, at best, to properly manage any development effort without the use of some type of metric. Software metrics can give us insight about the progress, quality, and expected completion of a software development effort.

In earlier software development efforts, programming was viewed as a "black art" and, consequently, software metrics were not commonly used. Today, it is generally accepted that a software development effort should be properly planned and that software metrics should be used to control the project. Program managers are no longer concerned about whether or not to use metrics but are more concerned with which metrics to use and whether or not the ones chosen will be effective. This question can best be answered by studying past uses of software metrics.

C. RESEARCH QUESTIONS

1. Primary Research Question:

What steps can a program manager take to ensure that the right software metrics are chosen and that the chosen metrics are effective?

2. Secondary Research Questions:

a. What are software metrics and what are they used for?

b. What is the Army policy on software metrics?

c. What software metrics are used in the Bradley A3 software development?

d. In what ways are the Bradley A3 software metrics effective in measuring program progress?

e. What lessons can be learned from the use of software metrics in the Bradley A3 program?

f. How can these lessons learned be generalized to guide and support other program managers?

D. SCOPE OF THESIS

This research uses the case of the Bradley A3 program to address issues concerning software metrics from a program manager's perspective. The software metrics used in the Bradley A3 program are analyzed based on their effectiveness in providing usable insight into the software development effort of this software-intensive weapon system.

E. METHODOLOGY

In order to provide a better understanding of the Bradley weapon system and the issues involved with software metrics, this research paper first provides a general overview of both the Bradley Fighting Vehicle and Software Metrics. In order to accomplish this I utilized the following resources:

- Department of Defense Publications
- Books, Periodicals, Journals, and electronic resources available at the Naval Postgraduate School (NPS) Library
- Internet web-sites pertaining to the Bradley Fighting Vehicle or software development
- Interviews with systems management faculty at NPS

Then, I conduct an analysis of the software metrics applied in the case of the Bradley A3 Program. In addition to the resources listed above I attained program

information from the Bradley A3 Program Office and United Defense Limited Partnership (UDLP), the primary contractor. Also, Government and contractor personnel with key roles in the Bradley A3 software development effort were interviewed. The end result of this case analysis is a group of lessons learned in applying software metrics in a software-intensive weapon system.

F. ORGANIZATION

In Chapter II, I provide an overview of the Bradley Fighting Vehicle. This includes a description of the vehicle and its role on the battlefield. Also, I describe the variants of the Bradley Fighting Vehicle and their differences with special emphasis on the A3 model. I conclude the chapter with a chapter summary.

In Chapter III, I provide an overview of software metrics. I begin by discussing what software metrics are and how they have evolved. Then I give examples of software metrics used. I then discuss the DoD policy on software metrics and conclude with a chapter summary.

In Chapter IV, I discuss how software metrics were applied in the Bradley A3 program. This includes a discussion of which software metrics were used and an analysis of how effective they were. Then I discuss the lessons that can be learned from this application of software metrics and how they can be generalized so that they will be useful to a program manager of any software-intensive program.

Chapter V is the last chapter of this thesis. I summarize the findings of this research and the answers to my research questions. Then I conclude with recommendations for further study.

G. BENEFITS OF STUDY

This research is the first at NPS addressing software metrics. The results of this research will provide valuable insight into the complexity of managing software-intensive weapon systems. Specifically, the lessons learned from the use of software metrics in the case of the Bradley A3 will make students at NPS and other future program/project managers in DoD more aware of how to apply software metrics to maximize their effectiveness.

II. THE BRADLEY FIGHTING VEHICLE

A. INTRODUCTION

The purpose of this chapter is to provide an overview of the Bradley Fighting Vehicle. I will first provide a vehicle description and discuss the role of the Bradley on the battlefield. Then I will describe the Bradley variants and their differences with special emphasis on the Bradley A3. I will conclude the chapter with a chapter summary.

B. DESCRIPTION

The Bradley Fighting Vehicle is a lightly-armored, fully-tracked vehicle that has a three-man crew and can carry six additional soldiers. It has a turret which provides fire control for the 25mm main gun, the TOW (Tube-launched, Optically-tracked, Wire-guided) missile, and the 7.62 mm coaxial machine gun. The 25mm main gun is the model 242 Bushmaster Chain Gun made by McDonnell Douglas Helicopter Company. It can fire in single-shot and multiple-shot modes and has dual-feed which allows the gunner to select from two different types of ammunition (typically HE-High Explosive and AP-Armor Piercing). The hull and turret of the Bradley are constructed of welded ballistic aluminum and have additional steel armor plates in the later models. The Bradley weighs approximately 67,000 pounds (combat loaded) and can travel 38 MPH on roads and 4 MPH in water. [Ref. 3, 14, 36]

C. THE ROLE OF THE BRADLEY

The Bradley Fighting Vehicle System includes two vehicles, the M2 Infantry Fighting Vehicle and the M3 Cavalry Fighting Vehicle.

The role of the Bradley M2 Infantry Fighting Vehicle is twofold. First of all, the Bradley provides the infantryman with additional firepower to destroy or suppress enemy tanks, vehicles, and troops. Secondly, the Bradley provides the infantryman with cross-country mobility to critical locations on the battlefield while protecting him from artillery and small arms threats. The Bradley M3 Cavalry Fighting Vehicle performs cavalry scout missions and carries three crew members plus two scouts [Ref. 3].

D. THE BRADLEY VARIANTS (A0 – A2 ODS)

There are four Bradley Variants in existence today. They are the A0, A1, A2 and A2 ODS (Operation Desert Storm). In this section I will briefly describe each of the variants and the new features that each one introduced.

1. Bradley A0

The First Bradley Fighting Vehicle A0 was fielded in 1982 as a replacement for the M113 Armored Personnel Carrier. It represented a vast improvement in capability since the M113's only armament was a .50 cal machine gun with no fire control and the M113 could not keep up with the M1 Tank in any terrain. This initial Bradley was armed with the 25mm cannon and the basic TOW missile. A total of 2,300 A0 Bradleys were produced from 1982 to 1986 [Ref. 3]. Out of the 2,300 A0 Bradleys produced, 510 were upgraded to A2 Bradleys. There are currently 1,744 A0 Bradleys in the Army inventory. [Ref. 34]

2. Bradley A1

In 1986 the Bradley Fighting Vehicle A1 was fielded. This new Bradley was equipped to fire the more lethal TOW 2 missile and was equipped with the final drives

used on the Multiple-Launch-Rocket-System (MLRS) which are more reliable. Also, vehicle and soldier survivability was improved with the addition of a revised fire suppression system and a gas particulate filter unit for use in chemically contaminated areas. Lastly, the equipment stowage in the vehicle was revised. A total of 1,371 A1 Bradleys were produced from 1986 to 1988 [Ref. 36]. All of these vehicles have been upgraded to A2 and A2 ODS Bradleys [Ref. 34].

3. Bradley A2

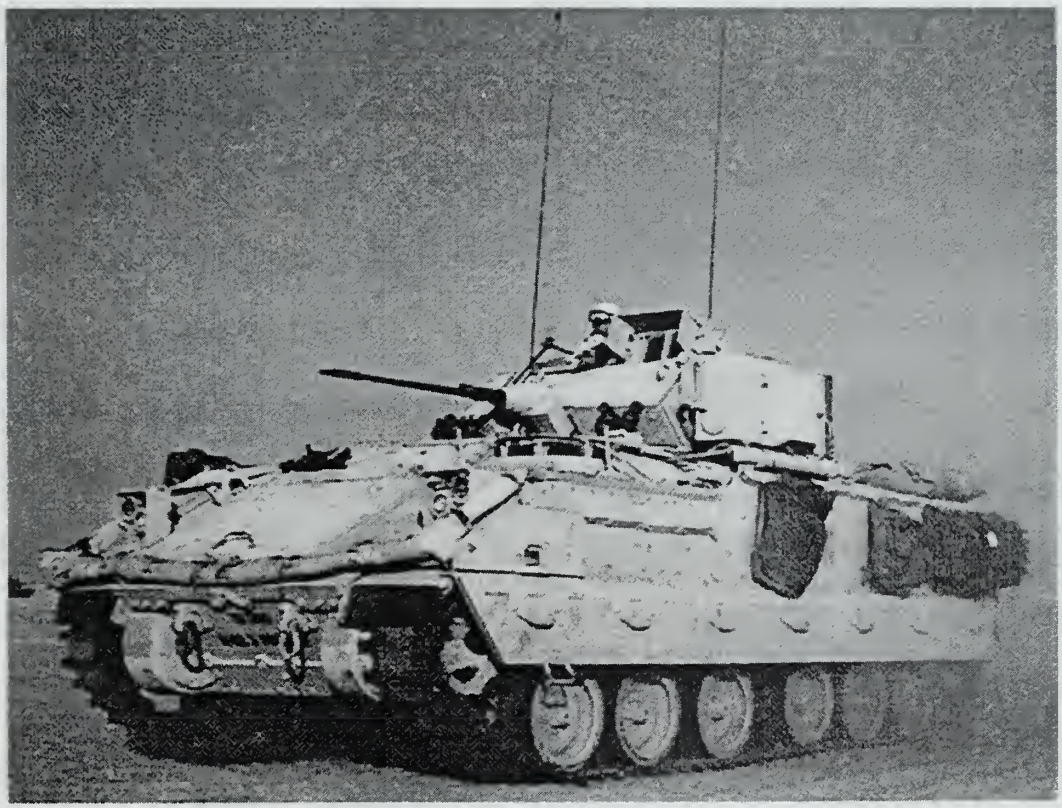

Fig. 2-1, Bradley Fighting Vehicle M2A2 During Desert Storm [Ref. 3]

In 1988 the Bradley Fighting Vehicle A2 was fielded. The A2 Bradley survivability was significantly better that that of previous Bradleys through the addition of steel armor plates on the exterior of the hull and turret. These steel plates provided

armor protection against munitions 30mm and smaller. Additionally, spall liners were added to the interior of the vehicle to reduce shrapnel in the event that a round penetrates the vehicle. This vehicle was also configured so that reactive armor tiles could easily be installed on the exterior of the hull and turret. To maintain the previous mobility capabilities of the Bradley, despite the additional weight of the armor steel plates and spall liners, a more powerful 600 horsepower engine and improved transmission were also included in the A2 Bradley. Lastly, the ammunition storage inside the vehicle was revised. Since 1988 3,107 new A2 Bradleys were produced and 1,356 A1 Bradleys and 510 A0 Bradleys were upgraded for a total of total of 4,973 A2 Bradleys [Ref. 36]. Of these 4,973 A2 Bradleys, 1,433 are being upgraded to A2 ODS Bradleys and 1,602 will be upgraded to A3 Bradleys, leaving 1,886 A2 Bradleys in the future Army inventory [Ref. 34].

4. Bradley A2 ODS

As with all major conflicts that the U.S. military is involved in, Operation Desert Storm was a catalyst for the emergence of new weapon systems and the rapid infusion of technology into existing systems. The Bradley Fighting Vehicle was one of the weapon systems which benefited from wartime-induced improvements through the introduction of the A2 ODS Bradley. The improvements on the A2 ODS Bradley are modular installations to increase specific capabilites and are not electronically integrated into the basic A2 Bradley configuration. Therefore, the improvements mentioned below can be added to existing A2 Bradleys as needed. The A2 ODS Bradley, which has not yet been fielded, includes a laser range finder that greatly improves the gunners' ability to get a

first round hit on target. It also includes a Global Positioning System / Position Navigation (GPS/POS NAV) system for navigation and a thermal viewer for the driver which increases nighttime mobility and safety. Lastly, the A2 ODS Bradley incorporates a combat identification system designed to decrease fratricide due to poor vehicle identification and a counter-missile device which is designed to electronically jam an incoming anti-tank missile. There will be 1,433 A2 Bradleys upgraded to the A2 ODS Bradley configuration [Ref. 34 and 36].

E. THE BRADLEY A3

The Bradley A3 represents a leap in technology when compared to its predecessors. It uses the same chassis and turret as the Bradley A2 and has the same general appearance, however, the similarities end there. The foundation for most of the improvements in the Bradley A3 is in the electronic design. The electronics in the older Bradleys are analog and are not integrated while the electronics in the Bradley A3 are digital and are fully-integrated through the use of two central processors and two 1553B Data Buses [Ref. 34]. These two processors and data buses not only provide the key to digital integration but also provide the system with redundancy which increases the survivability and reliability of the system. This electronic "backbone" allows the use of software to integrate new technologies and enhance the overall effectiveness of the Bradley A3 [Ref. 11, 22, 39].

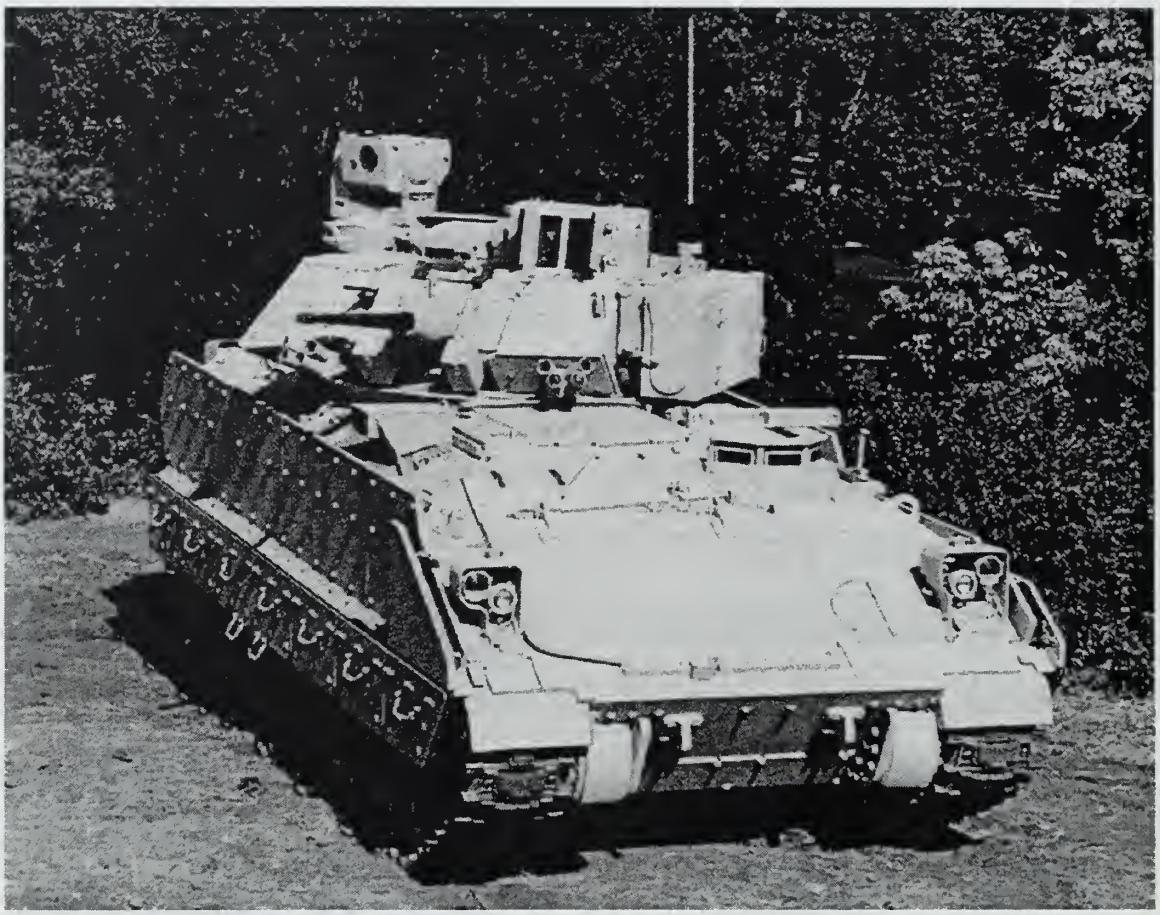

Fig. 2-2, Bradley Fighting Vehicle M2A3 [Ref. 9]

The Bradley A3 has numerous new capabilities. I will describe each of the Bradleys' major improvements by category (Command and Control, Lethality, Survivability, Mobility, and Sustainability).

1. Command and Control

There are many new technologies in the Bradley A3 which significantly increase the situational awareness of the vehicle commander and the dismount leader. The commander will have a flat panel display that depicts maps, operational graphics, and other tactical information. This tactical display will allow the commander to constantly be aware of the tactical situation without having to be distracted by large, unwieldy paper

maps. The Bradley A3 will also have a position/navigation (Pos/Nav) system with inertial navigation and Precision Lightweight GPS Receiver (PLGR) capabilities. The inertial navigation component of this Pos/Nav system provides constant location accuracy when the PLGR is not tracking sufficient satellites. This system is fully integrated with the commander's tactical display and the drivers' Pos/Nav display. In the Bradley A3, the commander now has his own independent viewer with 2nd Generation Forward Looking Infrared Radar (FLIR) and Day TV. This independent viewer allows the commander to scan a different area than the gunner, resulting in increased situational awareness at the full range of the optics. To increase the situational awareness of the dismounts, the Bradley has a squad tactical display inside the vehicle. This squad tactical display allows the dismounts to view the commanders' tactical display or whatever the gunner or commander are viewing through their respective optics. This display will also decrease the disorientation that the dismounts typically experience when exiting an enclosed mechanized vehicle [Ref. 8, 38, 39].

2. Lethality

The lethality improvements to the Bradley A3 are all part of the Improved Bradley Acquisition System (IBAS). Unlike the fire control on the previous Bradleys, the IBAS system has an integrated Laser Rangefinder so the gunner does not have to estimate range when shooting a target. The IBAS system also has Automatic Target Superelevation and Lead that use range, tracking, and ammunition ballistic data to automatically adjust the sight reticle for the ammunition being used and the range and motion of the target. These features decrease engagement time and vastly increase the probability of a first shot

hitting the target. The IBAS system also includes the Commanders' Independent Viewer (CIV) which was mentioned in the previous section. The CIV not only increases situational awareness but also increases the lethality of the weapon system. With the addition of the CIV, the Bradley crew can track two targets, to include tracking a second target while the first one is being engaged. Once the first target has been destroyed, the turret can be adjusted to engage the second target (being tracked with the CIV) with the flip of a switch. These Auto Dual Target Tracking and Auto Gun Target Adjustment features can be used with the 25mm main gun or the TOW missiles for each engagement. Lastly, the gunners' and commanders' viewers both use 2nd Generation FLIR that has greater resolution and clarity than night vision viewers on previous Bradleys. This improvement will improve the crews' ability to detect targets and correctly identify them, resulting in increased lethality and decreased probability of fratricide [Ref. 8, 38, 39].

3. Survivability

The Bradley A3 has two major improvements with respect to survivability. The first improvement is additional roof armor that improves protection against fragments from artillery rounds bursting above the vehicle. The second improvement is the addition of Gas Particulate Filter Unit (GPFU) and ventilation face pieces for the dismount element, which were previously only available for the vehicle crew. The GPFU and ventilation face pieces increase the soldiers' survivability when in an area contaminated by chemical weapons or nuclear fallout [Ref. 8, 38, 39].

4. Mobility

Like the A2 ODS variant of the Bradley, the A3 is equipped with the more powerful 600 horsepower engine and improved transmission. The Bradley A3 mobility is also improved through the addition of the all-weather drivers' viewer enhancer. This improvement is most evident when driving the vehicle during hours of darkness or other times of limited visibility (smoke, dust, or fog) [Ref. 8, 38, 39].

5. Sustainability

The sustainability of the Bradley A3 is improved through the integration of Built-In-Test (BIT) diagnostics. Since the entire weapon system is integrated through the 1553B Data Buses it is possible to get feedback on how each subsystem is operating. This diagnostic system conducts several diagnostic tests during regular startup and operation of the system and also provides the option to manually initiate a specific test. Previous Bradleys had separate test equipment that was bulky, difficult to use, and could not consistently isolate an electronic fault. Also, using this separate test equipment involved disconnecting and reconnecting several multi-pin cables that are delicate and easily damaged. Because the BIT diagnostics on the Bradley A3 are embedded in the vehicle, it will be easier to use and will decrease mechanic-induced faults [Ref. 8, 38, 39].

F. CHAPTER SUMMARY

Bradleys A0 through A2 ODS have provided us with capabilities which far surpass those of the M113 it replaced. These capabilities were validated by the overwhelming success of Bradley units in Desert Storm. However, it is also evident that the Bradley A3 will raise these capabilities to another level. The Bradley A3 has

capitalized on the strengths of previous Bradleys through the addition of key technologies that provide the system with enhanced capabilities. These enhanced capabilities, when combined into a fully-integrated digital system, will make the Bradley A3 the most capable mechanized vehicle ever produced. This increase in capabilities, however, has caused a phenomenal increase in the complexity of producing the weapon system. Every key technology added is dependent on software to operate as intended. Then, an even greater amount of software is required to combine each of these technologies into a fully-integrated system. The Bradley A3 is clearly a software-intensive system and, as a result, the success of the software development effort is closely related to the success of the overall system. This relationship between software and program success illustrates the importance of properly managing the software development effort through the use of software metrics.

III. SOFTWARE METRICS

A. WHAT ARE METRICS?

Software metrics are the combination of a specific measurement and its relationship to an established standard or index. Specifically:

> A software measurement is a quantifiable dimension, attribute, or amount of any aspect of a software program, product, or process. It is the raw data which identify various elements of the software process and product. Metrics are computed from measures. They are quantifiable indices used to compare software products, processes, or projects or to predict their outcomes. [Ref. 32, Ch. 8, p. 3]

Without measurement and metrics it would be impossible to truly manage software development. In other words "If you ain't measurin,' you ain't managin' — you're only along for the ride (downhill)!" [Ref. 32, Ch. 8, p. 2]. Measurement and metrics allow the manager to assess the status of his program to determine if it is in trouble, in need of corrective action, or process improvement [Ref. 32, Ch. 8, p. 1].

B. HISTORY AND EVOLUTION OF METRICS

Galileo (1564-1642) once said, "What is not measurable make measurable" [Ref. 19, p. 6]. This statement seems especially relevant to the field of software metrics. Through the use of software metrics, we have essentially measured what was once viewed as something you could not measure. Although the belief that software could be measured has only taken hold in the last decade, the field of software metrics is not a recent development. As early as 1968, R. J. Rubey and R. D. Hartwick published a paper on metrics titled "Quantitative Measurement of Program Quality" in Proceedings of the ACM National Conference [Ref. 10, p. 16]. Despite this early attention to the idea of

measuring program characteristics, programming was still viewed as a "black art". This view of programming was widely accepted for many years but the drawbacks of this view did not go unnoticed. Managers of software development efforts felt out of control since "You can't manage what you can't measure!" [Ref. 15, p. 3]. Also, the Department of Defense, which funded a large portion of early software development, was not comfortable with this total lack of control. As the use of computers and software increased, many began to question the view that programming was a "black art" and did not agree with the assumption that it was impossible to control software development efforts. Then, as the costs associated with software development began to increase [Ref. 5, p. 486], many began to ask: "Is it possible to identify or define indices of merit that can support quantitative comparisons and evaluations of software and of the processes associated with its design, development, use, maintenance, and evolution?" [Ref. 27, Preface]. By the mid-70s more attention was being given to the ideas of software metrics. It became evident to many that software measurement and metrics had the potential to make software development "concepts more visible and therefore more understandable and controllable" [Ref. 19, pp. 6-7]. Also, the number of articles and books written on the subject of software metrics increased dramatically. People began to change their view of software development as a "black art" and began to see that it had the potential to become more of a science. At the same time, the field of software engineering was in its early stages and gave birth to the idea that software could be designed in an organized and methodical manner. These changes would cause the use of software metrics to increase even more. However, by the early 80s, software metrics still were not widely-accepted

and those attempting to use them found that they were not always effective. In 1981, a Department of the Navy-funded research group concluded that there was "a great need for quantitative software measures, both for management and technical reasons, and that adequate measurement techniques did not exist" [Ref. 27, Preface]. One reason why the software industry was slow to embrace the use software metrics was that implementing a software metrics program required considerable effort and additional cost. Also, implementing a software metrics program did not guarantee that the metrics would tell the managers or customers what they were trying to find out about the project. There was no guarantee of success since software developers were still trying to understand some of the techniques of measuring software necessary for metrics to be effective. At the same time, the field of software development was moving quickly and new languages were continually being developed, making previously effective metrics less effective or obsolete. Finally, by the late 80s and early 90s the use of software metrics was generally accepted as a critical tool in the effective management of software projects. Capers Jones, a well-respected authority in the field of software metrics, makes this point in 1991.

> Measurement is the key to progress in software...Now that accurate measurements and metrics are available, it can be asserted that software engineering is ready to take its place beside the older engineering disciplines as a true profession, rather than an art or craft as it has been for so long. [Ref. 32, Ch. 8, p. 3]

Today, despite the industry-wide acceptance of the use of metrics, implementing them continues to be a significant challenge. Software projects continue to have problems and in some cases fail. This is especially evident by the number of General Accounting Office reports outlining the software problems experienced by defense related programs

in recent years. Norman E. Fenton and Shari L. Pfleeger identify some common problems with software projects in their 1997 book Software Metrics: A Rigorous and Practical Approach:

> It is difficult to imagine electrical, mechanical, and civil engineering without a central role for measurement. Indeed, science and engineering can be neither effective nor practical without measurement. But measurement has been considered a luxury in software engineering. For most development projects:
>
> -We fail to set measurable targets for our software products.
>
> -We fail to understand and quantify the component costs of software projects.
>
> -We do not quantify or predict the quality of the products we produce.
>
> -We allow anecdotal evidence to convince us to try yet another revolutionary new development technology, without doing a carefully controlled study to determine if the technology is efficient and effective [Ref. 19, p. 10]

Many of the problems software projects face may be due to larger planning issues; however, the proper use of metrics is almost always a common thread. Either metrics are not used at all or they are improperly used. A fundamental problem is that, without the proper use of metrics, a manager cannot predict cost or schedule or tell when ongoing software development is in trouble.

C. TYPES OF METRICS

Because software is developed for a wide variety of applications and is written in many different software languages, it is important to recognize that there are no "one size fits all" metrics. Software development program managers should be given the flexibility to determine which metrics are most useful for each specific project. Also, "quality, not quantity, should be the guiding factor in selecting metrics" [Ref. 32, Ch. 8, p. 27]. Since

every software development effort has its distinct characteristics, the variety of different metrics that could be used is seemingly infinite. In order to provide the background necessary for a basic understanding of metrics, only a few metric types will be discussed. I have chosen nine metric types that are fairly easy to understand and which address typical management issues for software development:

1. Size
2. Quality
3. Complexity
4. Requirements
5. Effort
6. Productivity
7. Cost and Schedule
8. Scrap and Rework
9. Support

In the following sections I briefly explain each of these metric types and give examples for each type.

1. Size

When discussing software, typically size metrics are the first type of metrics that come to mind. This is largely because the Source Lines of Code (SLOC) size metric was very popular when the use of metrics first began. The primary advantages of the SLOC metric are that it is relatively easy to measure and it is useful for cost/schedule models. The primary disadvantage of the SLOC metric is that it is difficult to estimate the total number of SLOC using requirement statements. Estimates for SLOC prior to development are typically done through analogy; comparing the current project with a similar past project. The accuracy of the estimate will be directly related to the similarity of the two projects. In DoD, where we keep systems for long periods of time, the two

projects being compared may not be similar enough to result in an accurate estimate. Also, the SLOC size metric can be difficult to use when mixing languages (in the current development or when estimating through analogy) since different languages can require varying SLOC to deliver the same functionality. Although SLOC is still being used, many software projects have switched to the use of Function Points as a size metric. Function Points are the weighted sums of five different factors that relate to user requirements: External Inputs, External Outputs, External Inquiries, External Files (interfaces to other systems), and Internal Files [Ref. 1, p. 639-640]. As the definition implies, the function points of a software project can be estimated using the software requirement statements. Another advantage of the function points metric is that it can be easily applied to higher level languages. Also, once estimates for function points are calculated, they can be used to estimate SLOC, which can then be used for cost/schedule models as mentioned earlier. The main disadvantage of the function points metric is that there are not extensive databases on the use of function points. Therefore, it is not possible to estimate project size through analogy using the function points metric [Ref. 32, Ch. 8, p. 36]. A side-by-side comparison of SLOC and Function Points is illustrated in Figure 3-1 on the following page.

2. Quality

Software metrics for quality can be defined in two ways: qualitatively and quantitatively. Many define software quality in terms of user satisfaction. This definition of quality can also be interpreted in many different ways. For example, you can measure the software in terms of user satisfaction for different attributes such as performance,

FUNCTION POINTS	SOURCE LINES-OF-CODE
Specification-based	Analogy-based
Language independent	Language dependent
User-oriented	Design-oriented
Variations a function of counting conventions	Variations a function of languages
Expandable to source lines-of-code	Convertable to function points

Fig. 3-1, Comparison of Function Points and SLOC [Ref. 32, Ch. 8, p. 33]

supportability, or operating cost. One can also measure software quality in terms of defects in the code. Again, this category can be broken down further into types of defects with respect to specifications, reliability, or survivability. [Ref. 32, Ch. 8, pp. 28-30] Figure 3-2 on the following page gives more examples of quantitative and qualitative metrics for quality.

3. Complexity

The complexity metric is relevant to software development because it is closely related to design errors and defects. Factors contributing to the complexity of software projects are size, interfaces among modules, and structure. The metrics for software size were already discussed. Common measures for interfaces among modules are "fan-in", the number of modules invoking a given application, and "fan-out", the number of modules invoked by a given application. Structure is simply the number of paths within a module. Some metrics use a combination of attributes, such as size, interface, structure, or other measures, to determine complexity while other metrics focus on only one software attribute. For example, the well-known McCabe's Cyclomatic Complexity

SOFTWARE QUALITY FACTOR	DEFINITION	CANDIDATE METRIC
Correctness	Extent to which the software conforms to specifications and standards	$Defects / LOC$
Efficiency	Relative extent to which a resource is utilized (i.e., storage, space, processing time, communication time)	$\frac{Actual\ Resourse Utilization}{Allocated\ Resourse Utilization}$
Expandability	Relative effort to increase software capability or performance by enhancing current functions or by adding new functions or data	$\frac{EffortToExpand}{EffortToDevelop}$
Flexibility	Ease of effort for changing software missions, functions, or data to satisfy other requirements	$(0.05)[AvgLaborDaysToChange]$
Integrity	Extent to which the software will perform without failure due to unauthorized access to the code or data	$Defects / LOC$
Interoperability	Relative effort to couple the software of one system to the software of another	$\frac{EffortToCouple}{EffortToDevelop}$
Maintainability	Ease of effort for locating and fixing a software failure within a specified time period	$(0.1)[AvgLaborDaysToFix]$
Portability	Relative effort to transport the software for use in another environment (hardware configuration, and/or software system environment)	$\frac{EffortToTransport}{EffortToDevelop}$
Reliability	Extent to which the software will perform without any failures within a specified time period	$Defects / LOC$
Reusability	Relative effort to convert a software component for use in another application	$\frac{EffortToConvert}{EffortToDevelop}$
Survivability	Extent to which the software will perform and support critical functions without failure within a specified time period when a portion of the system is inoperable	$Defects / LOC$
Usability	Relative effort for using software (training and operation, e.g., familiarization, input preparation, execution, output interpretation)	$\frac{LaborDaysToUse}{LaborYearsToDevelop}$
Verifiability	Relative effort to verify the specified software operation and performance	$\frac{EffortToVerify}{EffortToDevelop}$

Fig. 3-2, Example Metrics for Quality [Ref. 32, Ch. 8, p. 30]

Metric focuses specifically on the number of linearly independent paths (structure) through a program [Ref. 19, p. 293, 357]. Complexity metrics are typically collected through the use of automated tools: other programs designed to measure complexity attributes [Ref. 32, Ch. 8, pp. 38-39].

4. Requirements

A requirements metric is the number of requirements for a specific software project. The number of requirements is an important metric since many software projects have had problems or failed due to undefined, unclear, changing, or increasing requirements ("requirements creep"). If a requirements baseline is not established and monitored, then requirements will evolve simultaneously as the software is being developed, making previous efforts obsolete. Also, if requirements are not clarified early, then many unexpected implicit requirements will be discovered late in the development and cause problems. By tracking requirements metrics, the manager has visibility of both the existing requirements (explicit and implicit) and any deviations from the requirements baseline [Ref. 32, Ch. 8, p. 40].

5. Effort

An effort metric is the amount of effort required for a specific software development. An example measure used for effort is the total man-months required to complete the project. Past software development programs have observed that there is not a linear relationship between the effort and size of a project. In other words, you cannot assume that a project twice the size of a similar project will require twice the man-months to complete. Also, you cannot assume that by doubling the number of

programmers assigned to a project you can complete the project in half the time. In most cases, as the project size increases or the duration allowed decreases, there is an exponential increase in effort required [Ref. 32, Ch. 8, p. 41]. The phenomenon I just described was best explained by Frederick P. Brooks in his popular book "The Mythical Man-Month"; he attributed it to:

> Sequential Tasks - Many tasks in software development, such as debugging, must be completed in sequence. You cannot assign additional people to future tasks.
>
> Training – The training required increases when more programmers are involved. Also, development progress actually slows down if programmers are diverted to bring other programmers "up to speed" on the current development.
>
> Intercommunication – When more programmers are involved in a development effort, more pairwise intercommunication is required. In some cases, when too many programmers are assigned to a project (usually to save time), the effort of communication quickly dominates the individual effort on the development. [Ref. 7, pp. 44-52]

Typically, the effort required for a given project is estimated and then compared to effort expended. Estimates for effort are usually calculated based on decomposition (sum of the efforts for lower-level activities) or by using a model. Developers may also estimate using analogy (comparison to a past project) or expert opinion, however, these methods are not preferred. The effort, cost, and schedule metrics are closely related since the accuracy of the effort metric will have a direct impact on cost and schedule. In recognition of this relationship, some developers may group effort/cost as one metric. [Ref. 19, p. 435]

6. Productivity

Productivity metrics attempt to answer the question of how much actual software you get for a specific amount of effort expended. The most common measure used is the number of SLOC or Function Points per staff month [Ref. 19, p. 408]. The idea is that by knowing the productivity of a project you are more able to maximize it and more accurately estimate cost and schedule. Some of the common factors affecting productivity are in Figure 3-3. Items with larger numbers have a greater impact on productivity.

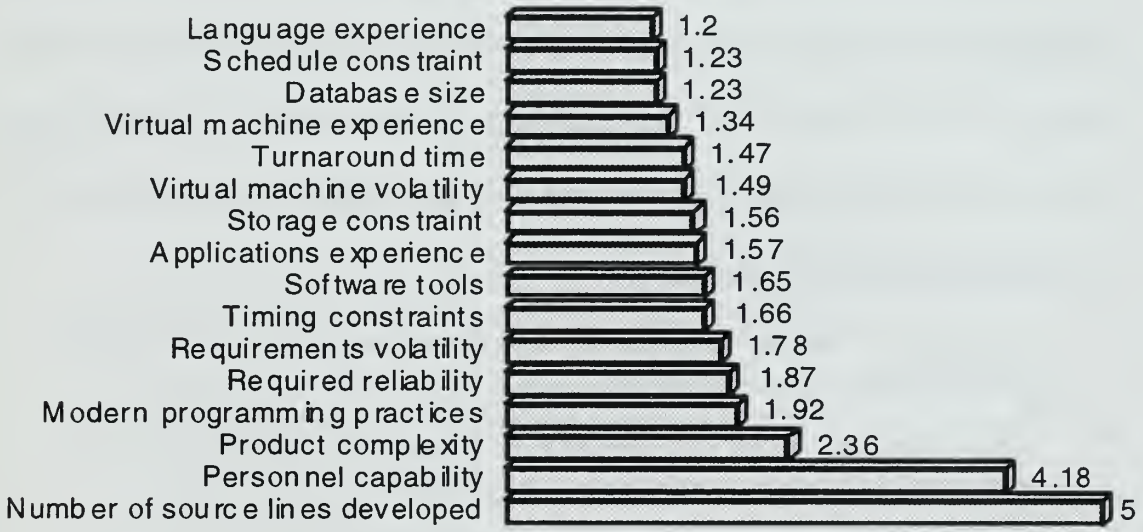

Fig. 3-3, Software Productivity Factors [Ref. 32, Ch. 8, p. 30]

7. Cost and Schedule

Cost and Schedule are probably the aspects of a software project that managers and customers are most interested in since they are measures that have traditionally been used for all types of projects. Also, cost and schedule help define the scope of a project and allow managers to track two very important resources: money and time. Although

size has the greatest impact on cost and schedule, there are other resources to consider, such as:

> Human resources – "The most significant resource of software development" (Boehm 81). Skills and experience can vary widely from person to person.
>
> Hardware resources – Development platform, that is, the computers used to write code. Also, compilers for specific software languages.
>
> Software resources – Special software tools used to develop or test the software code being developed.
>
> Reusable resources – Existing software modules which will be integrated into the current software development. [Ref. 32, Ch. 8, p. 45-46]

The actual measures for cost and schedule of a project are determined based on estimates. Because of the importance of these estimates, program managers typically use an estimation methodology which they feel best suits their project. Here is a brief overview of some commonly used estimation methods.

> Analogies – Cost and Schedule are determined base on data from similar past projects.
>
> Expert Opinion – Cost and Schedule are determined based on input from personnel with much experience on similar projects.
>
> Parametric – Cost and Schedule are determined based on automated parametric modeling tools which establish a relationship between "cost drivers" (independent variables) and the cost and schedule of the project (dependent variable).
>
> Engineering Build – Cost and Schedule are determined based on the sum of the estimated effort to complete each of the lower level tasks which constitute the entire project.
>
> Cost Performance Report Analysis – Cost and Schedule are determined based on current progress within the project.
>
> Cost Estimating Relationships – Cost and Schedule are determined based on a relationship between and independent variable, such as size, and the cost and schedule (dependent variable) of the project. Unlike Parametric

estimates only one independent variable is used and the process is not automated. [Ref. 32, Ch. 8, pp. 47-48]

8. Scrap and Rework

There has never been a programmer who could write "perfect" code. There will always be errors and defects when developing software. Many of these errors and defects will have to be fixed for the software to function properly. In most cases, fixing "bad" code will constitute a large portion of the overall development effort. Some early data suggest that code rework costs account for about 40% of all development expenditures. Unfortunately, there are no existing models to estimate the amount of scrap and rework a project will experience [Ref. 32, Ch. 8, p. 49]. Therefore, the program manager should reduce the risks associated with scrap and rework by:

- Using procedures to identify defects as early as possible;
- Examining the root causes of defects and introducing process improvements to reduce or eliminate future defects; and
- Developing incentives that reward contractors/developers for early and comprehensive defect detection and removal.
- Keeping track of the costs associated with scrap and rework to provide incentive for the software developer to emphasize proper planning, design, and other defect preventive measures. [Ref. 32, Ch. 8, pp. 48-49]

In addition to taking measures to reduce the risk of scrap and rework it is helpful to measure them as the development progresses. This will help minimize scrap and rework by bringing more attention to them. Also, having these measures will allow the developer to formulated parametric estimates (comparison based on project attributes) of scrap and rework for future phases of development and for future projects. Once the

developer has credible estimates, he will have useful scrap and rework metrics. [Ref. 32, Ch. 8, pp. 48-49]

9. Support

It has been determined that Post Deployment Software Support (PDSS) is a major contributor to the life-cycle cost of defense related software-intensive systems. This is because these software-intensive systems are continually upgraded to either fix existing problems or add new capabilities. By building a system with high software supportability we can decrease the life-cycle cost of the system and ensure that we can easily take advantage of the flexibility which software offers. Here are some of the metrics which can help a manager focus on software supportability:

Memory size - This metric tracks spare memory over time. The spare memory percentage should not go below the specification requirement.

Input/output - This metric tracks the amount of spare I/O capacity as a function of time. The capacity should not go below the specification requirement.

Throughput - This metric tracks the amount of throughput capacity as a function of time. The capacity should not go below specification requirements.

Average module size - This metric tracks the average module size as a function of time. The module size should not exceed the specification requirement.

Module complexity - This metric tracks the average complexity figure over time. The average complexity should not exceed the specification requirement.

Error rate - This metric tracks the number of errors compared to number of errors corrected over time. The difference between the two is the number of errors still open over time. This metric can be used as a value for tested software reliability in the environment for which it was designed.

Supportability - This metric tracks the average time required to correct a deficiency over time. The measure should either remain constant or the average time should decrease. A decreasing average time indicates supportability improvement.

Lines-of-code changed - This metric tracks the average lines-of-code changed per deficiency corrected when measured over time. The number should remain constant to show the complexity is not increasing and that ease of change is not being degraded. [Ref. 32, Ch. 8, pp. 50-51].

D. DOD POLICIES ON SOFTWARE METRICS – PAST AND PRESENT

The first DoD acquisition policies on the use of software metrics were in DoD regulations 5000.3-M-1 and 5000.3-M-3 dated 1986. These regulations specifically outlined items that the Test and Evaluation Master Plan (TEMP) should address with respect to software development [Ref. 2]. One of these items was to collect software metrics to evaluate software test results. Despite the existence of these policies, software metrics were not commonly used. There was a general lack of focus on software management issues within software-intensive programs which resulted in a long line of problem-plagued programs. The Defense Science Board Task Force and the Army's Software Test and Evaluation Panel (STEP) confirmed this lack of focus in 1987 and 1990 respectively [Ref. 2]. In conjunction with their findings, the STEP made a number of recommendations to the Army on how to fix the software problem. One of these recommendations was that the Army enforce a specific set of software metrics. In response to the STEP recommendations, the Army created DA Pamphlet 73-1; "Army Software Test and Evaluation Guidelines" dated 1992 [Ref. 2]. This pamphlet mandated that all programs with software-intensive systems use 12 specific metrics.

1. COST - Tracks software expenditures ($ spent vs. $ allocated).

2. SCHEDULE - Tracks progress vs. schedule (event/deliverable progress).

3. COMPUTER RESOURCE UTILIZATION - Tracks planned vs. actual size (% resource capacity utilized).

4. SOFTWARE ENGINEERING ENVIRONMENT - Rates developer's environment (developer's resources and software development process maturity).

5. REQUIREMENTS TRACEABILITY - Tracks requirements to code (% requirements traced to design, code, and test).

6. REQUIREMENTS STABILITY - Tracks changes to requirements (user/developer requirements changes and effects).

7. COMPLEXITY - Assesses code quality.

8. BREADTH OF TESTING - Tracks testing of requirements (% functions/requirements demonstrated).

9. DEPTH OF TESTING - Tracks testing of code (degree of testing).

10. FAULT PROFILES - Tracks open vs. closed anomalies (total faults, total number of faults resolved, and the amount of time faults are open).

11. RELIABILITY - Monitors potential downtime (software's contribution to mission failure).

12. DESIGN STABILITY - Tracks design changes and effects (changes to design, % design completion).

These metrics came to be known as the Army "STEP metrics". Specific guidance in DA Pamphlet 73-1 dictated that the 12 metrics would be defined by 242 specific data elements that would be reported in a specific format [Ref. 2]. Because of the rigid nature of these requirements, they were subject to much criticism within the acquisition community, especially among program managers who thought the policy was unnecessarily restrictive. With the onset of acquisition reform initiated by Secretary of Defense Perry, the requirements set forth by DA Pamphlet 73-1 were relaxed somewhat

in 1994. Program managers of software-intensive systems were still required to report the 12 "STEP metrics", however, they were given the flexibility to determine which data elements would define the 12 metrics. Although this new policy was less restrictive than the previous one, many acquisition reform advocates felt that program managers should have more flexibility to decide what metrics were most relevant for their program [Ref. 2]. Then, in March 1996 a new set of acquisition reform initiatives was set forth by the new DoD 5000 series. The DoD 5000.2 Regulation shifted the focus from a specific set of metrics to six management issues.

1. Schedule and Progress - regarding completion of program milestones, significant events, and individual work items.
2. Growth and Stability - regarding stability of required functionality or capability and the volume of software delivered to provide required capability.
3. Funding and Personnel Resources - regarding the balance between work to be performed and resources assigned and used.
4. Product Quality - regarding the ability of delivered product to support the user's need without failure, and problems and errors discovered during testing that result in the need for rework.
5. Software Development Performance - regarding the developer's productivity capabilities relative to program needs.
6. Technical Adequacy - regarding software reuse and use of approved standard data elements, and compliance with the DoD Joint Technical Architecture (JTA). [Ref. 17, App. V]

This change of focus was further solidified in September 1996 by a memo issued by the Army Director of Information Systems for Command, Control, Communications, and Computers (DISC4) and in December 1996 by a new DA Pamphlet 73-7 which superceded DA Pamphlet 73-1. Program Managers were now free to select those metrics

which they felt were most appropriate for their weapon system. However, DA Pamphlet 73-7 suggests 14 "Army Metrics" (12 "STEP" metrics plus 2 new ones) which could easily be used to fulfill the requirement to keep track of the six management issues. The policy I have just described is the policy which remains in force today with the exception that an additional management issue was added to the 5000.2 Regulation in March 1998: Program Success - regarding achievement of performance measures that are linked to strategic goals and objectives [Ref. 17, App. V].

E. CHAPTER SUMMARY

As software development has evolved, software metrics have become the primary tool available to control and manage software projects. However, the acceptance of metrics was slow to take place since software developers and others were slow to change their view of software development. Finally, after many years and many failed software development efforts, software development has come to be viewed as a science and the use of metrics has become widespread [Ref. 19, pp. 6-7, Ref. 24, p. 5]. Despite the acceptance of software metrics, developers still face substantial challenges when using them. They must decide what they want to know about their project, which metrics will provide that information, and how to best implement the metrics chosen. Also, as with any measure, there are shortcomings to the use of particular metrics, so software developers are continually challenged to improve their measurements and metrics.

Since the Department of Defense funded many early software development efforts, it played a major part in the development of software metrics. As with many DoD policies, the policies on software metrics have seemed to follow cyclical patterns from

little to strict control. Initially, software requirements existed only to support testing. Then, a specific set of metrics, which consisted of specific data elements, was required. Finally, program managers today can choose the metrics they feel are most appropriate as long as the metrics provide insight to specific management issues.

Although software metrics are accepted among software developers and the Department of Defense, software-intensive programs have continued to have problems with their software development efforts. This would lead us to believe that we can still benefit from the study of software metrics. As Norman E. Fenton stated "where we can already measure, we should be making our measurements better" [Ref. 19, p. 6].

IV. SOFTWARE METRICS AND THE BRADLEY A3 PROGRAM

A. INTRODUCTION

This chapter provides a discussion and analysis of how software metrics were applied in the Bradley A3 program. First, a description of the past and present metrics will be provided. Then, the focus will shift to program management-level issues relevant to the application of software metrics. The three primary issues addressed in this chapter are: 1) The Purpose of Metrics, 2) Implementing an Integrated Metrics Program, and 3) Which Metrics to Use. These issues were derived based on interviews with key personnel and program information attained from both the Bradley A3 Program Office and UDLP. The Bradley A3 Program is discussed and analyzed with respect to each of these issues. The chapter concludes with generalized lessons learned, which are relevant to any software-intensive program.

B. BRADLEY A3 METRICS – PAST AND PRESENT

Before software development had begun on the Bradley A3, the program manager recognized that the contractor had little experience developing real-time, embedded software systems. To alleviate the risk to the program, he solicited the help of the U. S. Army Missile Command (MICOM, now Aviation and Missile Command, AMCOM) Software Engineering Directorate (SED, now Battlefield Automation Directorate, BAD). The SED role would be to assist the contractor and program management office with all issues involved in managing software development for the Bradley A3. This would

include both advising the two organizations on all software-related issues and direct involvement with respect to software metrics. [Ref. 13, 22]

When software development began on the Bradley A3, the twelve Army STEP metrics were collected. As mentioned earlier, these twelve STEP metrics were mandated by Army policy and were also included in the Statement of Work (SOW) for the Bradley A3. The STEP metrics were collected by the Software Quality Assurance (SQA) organization within UDLP and were briefed during quarterly program reviews. Neither the contractor's software managers nor the government program manager understood or used these metrics. Also, there was very little involvement by the software developers within UDLP in collecting the metrics. Each time these metrics were briefed, the audience struggled to gain some meaning from them. The reliability and fault profile metrics consistently confused the audience. As a result, these metrics, although still collected in accordance with the mandate and SOW, were no longer briefed. [Ref. 12, 13, 22]

Other metrics, however, proved to be more useful. Despite the Government's primary focus on the STEP metrics, UDLP had been keeping data and estimates for Source-Lines-Of-Code (SLOC). This data would prove the value of metrics as the program approached the software development for the Low-Rate-Initial-Production (LRIP). Using the data and estimates for SLOC, SQA was able to calculate a crude metric for productivity. They used this metric to determine that the first major software build (version 1.0) would not be completed until twelve months after the scheduled LRIP In-Progress Review (IPR) date. Based on this information and advice from SED, the

program manager decided to focus the software development effort on only those items that were critical for LRIP. The result was that the software was released (Renamed version 1.0.1) in time to support LRIP requirements, not twelve months late. [Ref. 12, 13, 22]

As the program office began briefing software metrics to the office of the Assistant Secretary of the Army for Research, Development, and Acquisition (SARDA), the focus shifted from the STEP metrics to a new set of metrics. These metrics were:

Problem/Change Request (PCR) Status (priority and age)
Test coverage (Breadth and Depth)
Productivity
Fault Profile [Ref. 12, 13, 41]

These metrics were established by the program manager based on discussions with SARDA executives and were typically presented on 3-D bar charts which showed a considerable level of detail. The entire packet consisted of 35 slides. Although these metrics provided a lot of detail, they yielded little information which was useful to the program manager or the SARDA executives. Also, because of the level of detail in the charts, many questions were raised during briefings. The result was that much time was spent explaining what certain charts were attempting to depict about the program. After using these metrics for a year, the program manager decided it was time to revise them. He had become frustrated by their shortcomings and saw the need for metrics which could better predict the software development status. [Ref. 12, 13, 22]

The intent for the new metrics was to answer two fundamental questions that all program managers are interested in: 1) What is the status of my program? and 2) Are there any trends indicating that trouble lies ahead? In conjunction with the contractor,

SED and the program manager settled on three metrics which would best answer the questions above. The three metrics, which are still being used today, are:

Software Development Progress
Test Progress
High Urgency Problem/Change Report (PCR) Closeout [Ref. 12, 13, 41]

The Software Development metric consists of the number of software tasks complete versus the number of software tasks scheduled. A sample chart of this metric is depicted in Figure 4-1 below:

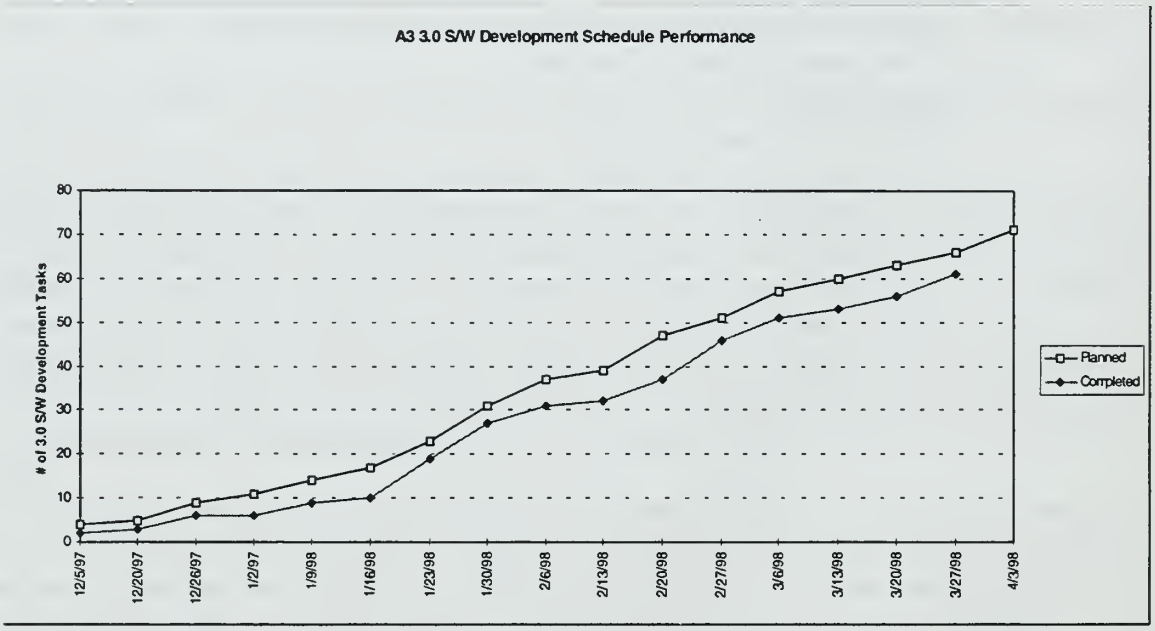

Fig. 4-1, Software Development Progress [Ref. 41]

Although this schedule metrics is very simple, it has provided the program manager with an accurate measure of progress [Ref. 12, 13, 22].

The Test Progress Metric consists of the number of testing tasks complete versus the number of testing tasks scheduled. A sample chart of this metric is depicted in Figure 4-2 on the following page:

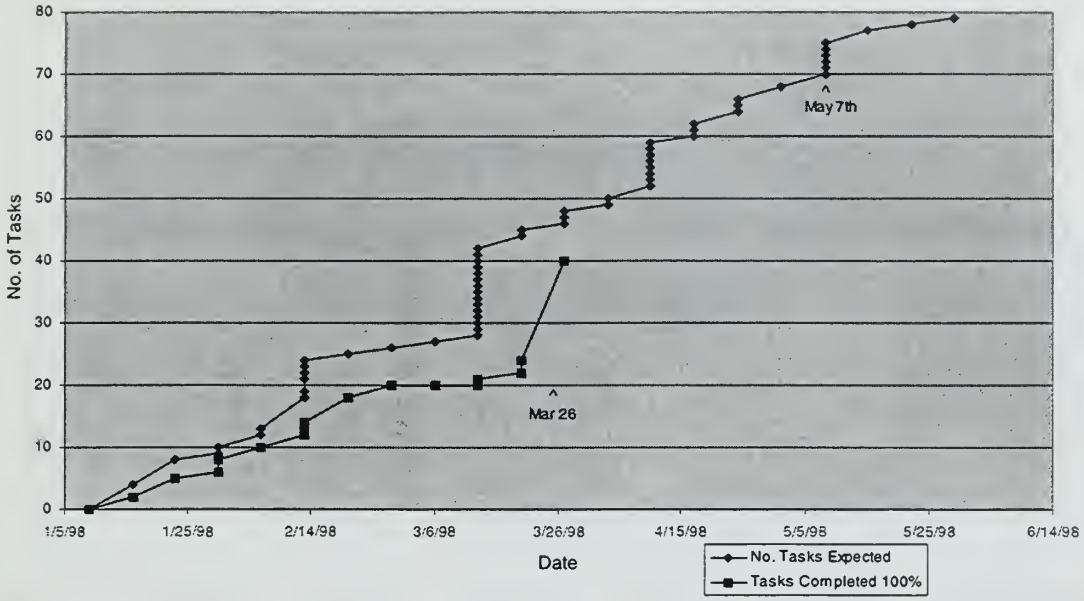

Fig. 4-2, Test Progress [Ref. 41]

The Test Progress Metric is another schedule metric but with a different perspective. The use of this metric reflects the program manager's recognition that there is more than one important aspect of program status with respect to progress. The actual software development (code writing) is the obvious indicator of program status, but unexpectedly slow progress in testing could just as easily cause schedule slips in the program.

The High Urgency Problem/Change Report (PCR) Closeout metric consists of the number of high urgency PCRs cancelled or closed versus the number of high urgency PCRs to be fixed in the current software build projected (based on resource constraints). High urgency PCRs are the most critical to fix since they are usually related to safety or mission-essential capabilities. A sample chart of this metric is depicted in Figure 4-3 on the following page.

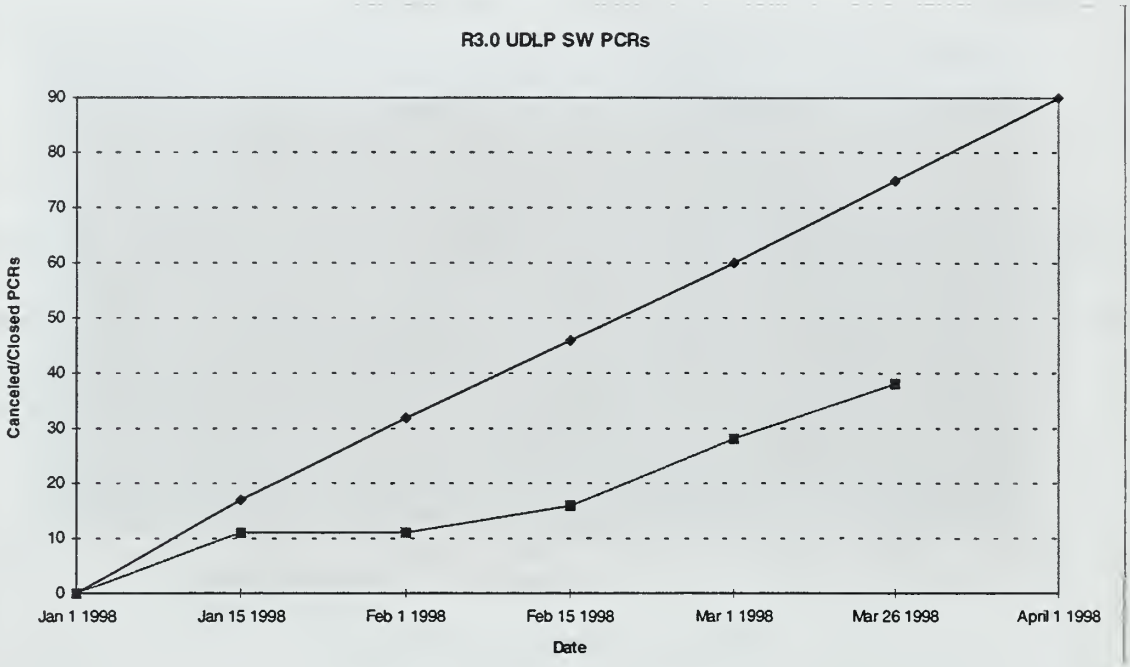

Fig. 4-3, High Urgency PCR Closeout [Ref. 41]

Note that the projected progress line on this metric is straight. In the future, historical data will be used to project the monthly rate of progress [Ref. 12, 13, 22].

In addition to the high priority PCR metric, the program manager is also briefed on the status of targeted PCRs. This chart depicts the current status of all PCRs that will be fixed in the current build. The targeted PCRs consist of key high and medium-urgency PCRs which must be fixed to meet project milestones for the entire Bradley A3 system [Ref. 12, 13, 22]. A sample of this chart is depicted in Figure 4-4 on the following page.

The PCR metric and status charts depicted above are important to the program manager for two reasons. First, they give the program manager insight into the quality of the software being developed. Second, they give the program manager insight into the software developer's ability to meet schedule requirements. This includes the

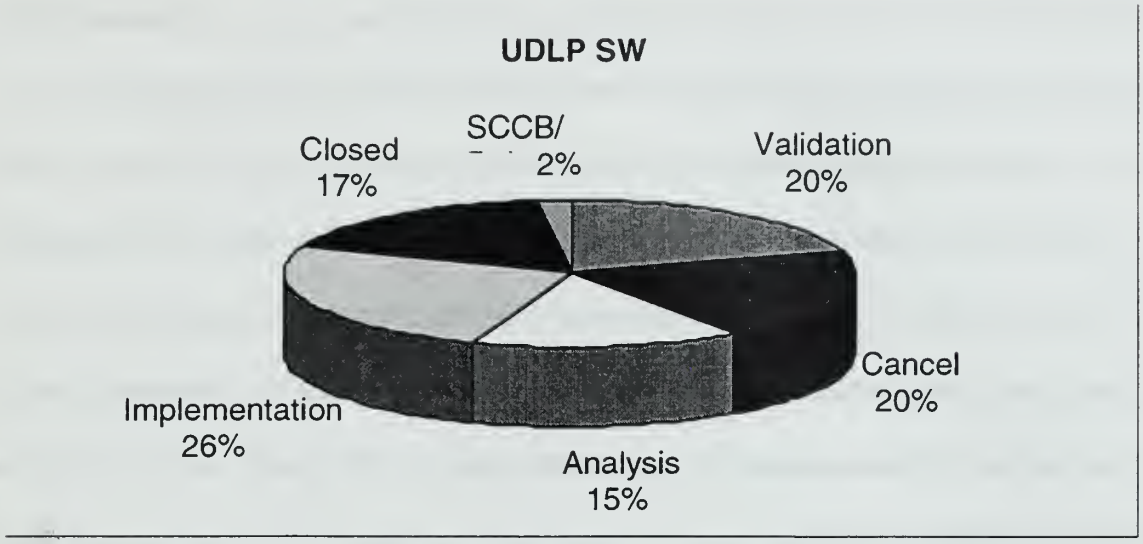

Fig. 4-4, Status of Targeted PCRs [Ref. 41]

consideration that focusing effort on fixing PCRs will have an impact on the contractor's ability to make progress on new software code.

The metrics I have described above have been very useful in enabling the program manager to stay informed on the status and trends of the software development. However, these are not the only metrics used in the software development effort for the Bradley A3. There are several other metrics which the UDLP software manager uses to ensure that every aspect of the software development effort is on track. [Ref. 12, 13, 22]

C. ANALYSIS OF KEY SOFTWARE METRICS ISSUES

1. The Purpose of the Metrics

The purpose of metrics in the Bradley A3 software development became an over-arching issue that the contractor and the program manager struggled with. This struggle began early in development when the STEP metrics were being used. These metrics did

not help the software manager or the program manager manage the program and did not have a meaningful purpose other than meeting the Army and SOW requirements.

The Army policy mandating the use of specific STEP metrics and component data elements caused the contractor and the program manager in the Bradley A3 program to focus on metrics as a requirement to be met. This is expected with any policy that tells an organization how to do something instead of telling them what they really want them to accomplish. This is analogous to our use of military specifications and standards where we would tell the contractor exactly what to build and, after he built it, we would realize that the product did not meet our needs. The Army policy makers' intent was to improve software acquisition management. However, the STEP policy did not focus on improving software management. Instead, it focused on collecting certain metrics and data elements. The assumption was that these metrics would provide the right insight into software programs and, as a result, improve software acquisition management. However, as the Bradley A3 Program has experienced, this assumption did not always hold true. The program and software managers never had to ask themselves what they wanted to know about the program with respect to software. The Army had dictated what aspects of the software development they would keep track of, so that is what they did. Finally, after realizing that the STEP metrics were not telling him anything useful about the program, the program manager changed the metrics. The new metrics were an improvement over the STEP metrics, however, SARDA executives had much influence on what they would consist of. Because of this, the focus was still somewhat on collecting specific metrics versus using metrics which would tell the program manager

what he needed to know about the program. Finally, after once again realizing that this set of metrics was not very useful, the program manager decided to change the metrics again. This time the program manager, with the help of SED, decided on the final set of metrics which are used today. [Ref. 12, 13, 22]

In addition to the problems with the Army policy on metrics, the contractor's limited experience with embedded software contributed to the lack of meaningful purpose for metrics. Since UDLP had never undertaken a software development of this scale, they had not used metrics enough to understand how important metrics would be to the program success. Also, they did not have an understanding of the valuable insights that metrics could provide so that the managers could effectively manage the software development. The result was that UDLP initially did not focus on their metrics program. It was viewed as a requirement to be met, not as an essential part of software management. [Ref. 12, 13, 22]

As with the contractor, the program manager also had limited experience with software development. It would be difficult for anyone with limited experience to understand the complexity of the software development and the importance of software metrics. Also, a manager with limited exposure to programs that had used metrics effectively would more easily lose confidence in metrics. This was especially relevant during the early software development for the Bradley A3 because of the difficulties discussed above. As with the contractor, the result of the program manager's limited experience was that the purpose of the software metrics was not focused on managing the program, but was seen as requirement to be met.

2. Implementing an Integrated Metrics Program

The implementation of the metrics program in the Bradley A3 development could best be described as a learning process. The contractor, having had little experience with software metrics, was not quite sure how to implement a metrics program. More importantly, they did not understand the utility of implementing an effective program. The result was that the metrics were not very useful until later in the development effort.

Early in the software development for the Bradley A3, the metrics program was not an integrated part of the software development. When UDLP began the software development, they had their Software Quality Assurance (SQA) organization collect the data for the STEP metrics. By doing this, they had essentially separated the development effort from the metrics program. The software developers were responsible only for the development, not for the metrics which were supposed to be tracking the development. The separation between the software development and metrics is a clear indicator that the metrics were not being used as a management tool by the contractor. As mentioned in the previous section, the requirement for metrics was being met, but they were not assisting the managers (Government and contractor) in managing the program. The effectiveness of metrics is based on the extent that they provide useful information to the managers so that they can better manage the development effort. Because of the separation between software development and metrics during implementation, the early metrics used in the Bradley A3 program were not effective [Ref. 12, 13, 22].

As the software development for the Bradley A3 progressed, the integration of the metrics program improved and the metrics became more useful. One of the driving

factors behind this improvement was the involvement of the MICOM SED. They played a key role in improving the use of metrics. First of all, through their interaction with the contractor, they were able to provide advice on the use of metrics and how to ensure that they were being used effectively. Through their expertise in software, they had a better vision of the insights metrics could provide the managers. The SED also provided a great deal of assistance to the program management office. They conducted a software manager's "short course" to educate the program management office on software-related issues, including metrics. The SED also provided the program manager with valuable advice on the use and implementation of software metrics. By patiently working with the contractor and by assisting them during implementation, the SED was able to help the metrics program become integrated with the software development.

3. Choosing Metrics and Ensuring Their Effectiveness

This section will focus on choosing the right metrics and methods for increasing their effectiveness. First, an analysis of the metrics used on the Bradley A3 Program will be provided. This will be followed by a discussion and analysis of how the Bradley A3 Program increased the effectiveness of their metrics by using two different sets of metrics tailored for the software management and program management levels. Then a discussion and analysis of the importance of metrics presentation will be provided. The section will close with a brief discussion and analysis of cost as a factor for choosing metrics.

As mentioned in previous sections, early in the Bradley A3 Program, the STEP metrics were not providing the software manager and program manager with useful

information for managing the program. Under the Army's STEP metric policy, choosing which metrics to use was not much of a consideration. Later, when the Army mandate for STEP metrics was removed and the PM had decided to change the metrics, choosing the right metrics became a relevant issue. Not being forced to use a specific set of metrics was a step in the right direction. While under the Army STEP metric policy, the contractor and program manager had begun to realize that metrics are not "one size fits all." The STEP metrics were not providing the program manager with right information that he needed to effectively manage the program. One of the main drawbacks of the STEP metrics was that they provided the program manager with too much detailed data and not enough useful information. The next set of metrics were chosen based on input from SARDA executives. These metrics were somewhat more useful to the program manager since they focused on broader management issues more appropriate to the program manager. However, like the STEP metrics, so much detail was provided that it was easy to become distracted by difficult to understand metrics and lose sight of the more important issues. One of the underlying problems with both the STEP metrics and this second set of metrics, was that they were not created by the manager to support the manager. They were created by those external to the program management office, in one case the Army, and in the other case SARDA executives. Finally, unrestrained by external influence, the program manager was able to independently decide what he wanted to know about the program. This is the first step toward choosing the right metrics for a particular project. Because of his broad perspective of the program, the program manager decided that he wanted to know about the status of the program and any

trends indicating future trouble. This was the first time that questions about the program preceded the formulation of metrics to be used. Given the program manager's questions, SED and the program manager decided that the three metrics below were most appropriate:

Software Development Progress
Test Progress
High Urgency Problem/Change Report (PCR) Closeout [Ref. 41]

Because these metrics were chosen specifically to answer the program manager's concerns about the program, they ended up being very effective.

The three metrics above are distinctly different than the metrics used by the software manager. This makes a great deal of sense since the software manager's perspective is very different than that of the program manager. The software manager will want to know detailed information about different aspects of the development while the program manager is more concerned about overall trends. Also, the software manager has a much deeper understanding of software development and metrics, so he is more likely to understand detailed metrics than a program manager. The disadvantage of the program manager using only a few, very broad, metrics is that he may lose visibility on obscure information which may indicate a problem in development. There are several things the program manager can do to overcome this shortcoming. First, he can establish a relationship of trust and cooperation with the contractor so that the contractor's software manager will "raise the red flag" when he becomes aware of a potential problem that is not reflected in the program manager's metrics. Second, the program manager can get educated on software development and metrics to the extent that he will be more aware of

software issues and the areas of the development that are most likely to cause problems. Last, the program manager can solicit the help of organizations external to the contractor with expertise in software development and metrics. This will provide the program manager with an independent set of experts to help him oversee the software development effort at a more detailed level without being overwhelmed by data he doesn't understand.

The presentation of the metrics chosen is also critical to their effectiveness. Even if the metrics answer the questions posed by the manager, they will not be useful if the manager cannot understand the way they are being presented. "The measurements must be presented in a way that tells both the customer and developer how the project is doing" [Ref. 19, p. 19]. Note the simplicity of the sample charts in Section B. These charts clearly depict the trends for each of the metrics and can be understood with very little explanation.

Another factor to consider when choosing software metrics is what you can afford. Like all reporting requirements imposed on a contractor, the collection of software metrics is not free. This is especially true in a cost type contract such as the one in place with UDLP. The program management office was fully aware of this issue when choosing metrics for the Bradley A3 program. They have decided to use only those metrics which are necessary to effectively manage the program. This would be the wisest decision for any program given the competition for limited resources in today's acquisition environment.

D. GENERALIZED LESSONS LEARNED

The following lessons learned were derived from the research conducted in this case analysis of the Bradley A3 metrics. However, they have been generalized for application to any software-intensive program.

1. Hire the Experts

The program manager of any software-intensive program should solicit the help of an independent agency to assist in overseeing software development. For example, it is seldom the case that the program manager has enough experience with software to fully understand all the issues involved with software development. Also, in many cases, the contractor is not very experienced in software development either. Having support from an agency with extensive experience in software development will mitigate the risk associated with having an inexperienced contractor. This is especially the case with complex, real-time, embedded systems.

2. Focus Metrics on Managing the Program

Having metrics which focus on the purpose of managing the software development effort is critical to metrics effectiveness. For example, metrics will not be effective if they are collected simply to meet a requirement or to appease an agency external to the development program. By ensuring that the purpose of the metrics is to fulfill the need to manage the software development, you will be providing the foundation for useful insight into the development.

3. Implement Only the Most Useful Metrics

The program manager should only implement the most useful metrics which are absolutely required to manage the program. For example, he must resist the temptation to try to know every detail of the software development. This is especially true in today's acquisition environment where there is great competition for limited resources. We simply cannot afford the luxury of unlimited information with respect to software development.

4. Make the Software Developer Responsible for Metrics

To ensure that metrics are effective, they must be fully-integrated with the software development effort. One way that the program manager and contractor can promote this integration is by ensuring that the software developer is also responsible for metrics. By being responsible for metrics, the software developer will not view metric collection and tracking as separate functions that do not matter. Ideally, the software developer will embrace the use of metrics and the information they provide managers at all levels.

5. Tailor Your Metrics (Management Level, Stage, and Presentation)

Metrics will be most effective if they are tailored to the specific application. Tailoring can take on many forms. First, metrics can be tailored for the specific level of management. For example, metrics that the program manager uses should be broad and depict general trends, while those that the software manager uses should be more detailed. Second, metrics should also be tailored for each stage of the development. For example, early in the software development it would make sense to focus on a requirements metric,

while later in the development it would make more sense to focus on a design stability metric. Last, the metrics presentation should be tailored for the specific audience. This is important since the effectiveness of a metric is based on the information it provides to the managers. A metric that is presented poorly or not understood will not be effective.

6. Get Educated on Software Development and Metrics

When managing any software-intensive system it is vital that the program managers have at least a general understanding of software-related issues. There are few opportunities for formal training and there is rarely excess time in any duty position. Therefore, getting educated on software development becomes a matter of personal development. There are organizations, such as the Software Program Manager's Network [Ref. 31] or the Army Software Metrics Program [Ref. 2] that can answer a wide variety of questions and recommend specific books or journals. Knowing more about software and metrics will help the program manager deal with the challenges associated with managing a software-intensive program. Also, it is much better to have that knowledge before becoming the program manager or before the software development begins. Although it is possible to learn much about software while managing its development, knowing more prior to the development will allow the program manager to take full advantage of that knowledge.

7. Cooperative Relationships Foster Success

When developing any system, having cooperative relationships with the contractor and other agencies involved greatly increases the probability of overcoming problems during development. Having a contentious "you vs. me" relationship will only exacerbate

the effect of problems during development. This has been observed in many defense-related programs. When a cooperative relationship exists, the contractor is not afraid to inform the program manager of a problem. Also, with a cooperative relationship, the response to a problem is more likely to focus on solving the problem, versus the contentious relationship, where the focus shifts on who's fault it is. This positive response to problems in cooperative relationships allows all parties involved to combine their respective resources and find appropriate solutions. Lastly, the effects of having cooperative relationships are multiplied when developing complex, software-intensive systems.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This research shows that, based on lessons learned from the Bradley A3 program, the most significant software metrics issues for a program manager are: focusing metrics on the purpose of managing the program, and choosing the best set of metrics to use.

Software metrics were first created in hopes that software development could be controlled and managed. As software development has evolved, developers have gained a better understanding of metrics and proven that metrics are valuable management tools. Using metrics with a focus on anything other than managing the program would result in a lack of clear direction for the metrics program. Without without a clear understanding of why you are using metrics, it is unlikely that the metrics program will be successful. Metrics cannot be used simply to fulfil a requirement and then expect that they will assist the manager in managing the development. In short, if managers want metrics to provide them with insight about software development, they must not lose sight of the original purpose of metrics: management.

Once the purpose of metrics is focused on managing the program, the next step is to choose which metrics to use. As with the purpose of a metrics program, choosing which metrics to use should be motivated by the needs of the manager. Using certain metrics because they are mandatory or because someone external to the program recommends them, will decrease the likelihood that the metrics will provide useful information. In most cases, using metrics because of external influence will cause the

program to expend resources gathering data which does not apply to the present effort and fail to gather data which could translate into useful information. Also, the metrics that the program manager uses do not need to be the same as those that the software manager uses. Because of their different levels of management, their information needs are different; therefore, the metrics they use should be different. Ideally, the original source of any metric should be in the form of a question. As the manager plans, he will have questions based on information he has identified that he will need to effectively manage the program. Then, specific metrics should be chosen to answer those questions.

B. RECOMMENDATIONS

There are many factors which influence the success of software metrics. Since program managers cannot influence every one of these factors, it would seem logical to influence those which he can and minimize the negative impact of the others.

The first factor, which the program manager has direct control over, is getting educated on software development and metrics. This would be the best way to prepare for managing a software-intensive system since it will have a direct impact on the program. As a key decision-maker, the program manager is in a position to influence the success of the development, including the use of metrics. As mentioned earlier, poor management has been cited as the primary cause of failures in software-intensive systems [Ref. 32, Ch. 1, p. 18]. The best way that we can prevent poor management from being the cause of failures in the future, is by getting educated today. A program manager who is more knowledgeable on software issues and metrics will be more capable at managing software-intensive systems. Since he will already have a good knowledge base about

software and metrics, he will be able to focus more on managing the program. He will have a better understanding of areas or phases of the development that may be more likely to encounter problems and will be better equipped to ask the right questions about these areas before undetected problems get out of control. Also, the informed program manager, based on his knowledge of metrics, will better understand the insight that they can give him in managing the program. With this understanding, he will see the need for effective metrics and will be more capable of choosing the right ones.

The second factor that the program manager has some control over is minimizing the impact of a contractor who is not experienced at developing software-intensive systems. This research has shown that the best way a program manager can accomplish this is by soliciting the help of an independent agency. This independent agency can assist the program manager in overseeing the contractor and assist the contractor in increasing his development capabilities. It is difficult for any organization to develop new capabilities without any assistance, especially if they are trying to utilize this new capability as it is being established. This is the case when we expect a contractor to attain the capability to develop software-intensive systems while actually developing one. This situation creates a great deal of risk for a program. By hiring an independent agency we can effectively mitigate this risk.

C. ANSWERS TO RESEARCH QUESTIONS

1. What are software metrics and what are they used for?

Software metrics are the combination of a specific measurement and its relationship to an established standard or index. Specifically:

> A software measurement is a quantifiable dimension, attribute, or amount of any aspect of a software program, product, or process. It is the raw data which identify various elements of the software process and product. Metrics are computed from measures. They are quantifiable indices used to compare software products, processes, or projects or to predict their outcomes. [Ref. 32, Ch. 8, p. 3]

Software metrics are used by program and software managers to manage software development. Measurement and metrics allow the manager to assess the status of his program to determine if it is in trouble, in need of corrective action, or requires process improvement [Ref. 32, Ch. 8, p. 1]. See section III.A. for more detail.

2. What is the Army policy on software metrics?

The Army policy on software metrics was changed in March 1996 by DoD 5000.2 Regulation, which shifted the focus from a specific set of metrics to six management issues. The regulation stated that program managers should implement metrics which address theses six management issues:

1. Schedule and Progress - regarding completion of program milestones, significant events, and individual work items.
2. Growth and Stability - regarding stability of required functionality or capability and the volume of software delivered to provide required capability.
3. Funding and Personnel Resources - regarding the balance between work to be performed and resources assigned and used.

4. Product Quality - regarding the ability of delivered product to support the user's need without failure, and problems and errors discovered during testing that result in the need for rework.

5. Software Development Performance - regarding the developer's productivity capabilities relative to program needs.

6. Technical Adequacy - regarding software reuse and use of approved standard data elements, and compliance with the DoD Joint Technical Architecture (JTA). [Ref. 17, App. V]

The policy I have just described is the policy which remains in force today with the exception that an additional management issue was added to the 5000.2 Regulation in March 1998: Program Success - regarding achievement of performance measures that are linked to strategic goals and objectives [Ref. 17, App. V]. See Section III.D. for a detailed description of past Army policies on software metrics.

3. What software metrics are used in the Bradley A3 software development?

Currently, the Bradley A3 software development (program management level) uses these three metrics:

Software Development Progress
Test Progress
High Urgency Problem/Change Report (PCR) Closeout [Ref. 12, 13, 41]

For more charts and description of these metrics see Section IV.B.

4. In what ways are the Bradley A3 software metrics effective in measuring program progress?

The Bradley A3 software metrics are effective because they provide the program manager with the information he needs to manage the program. In this case, the program manager decided that he needed to be informed about the overall software development progress, the test progress, and high urgency PCR closeouts. These three metrics allow

the program manager to assess the status of his program to determine if it is in trouble, in need of corrective action, or process improvement [Ref. 32, Ch. 8, p. 1]. Also, they are presented in an easily-understandable format. See Sections IV.B. and IV.C. for more detail.

5. What lessons can be learned from the use of software metrics in the Bradley A3 program?

The following lessons learned were derived from the research conducted in this case analysis of the Bradley A3 metrics. However, they have been generalized for application to any software-intensive program.

1. Hire the Experts - The program manager of any software-intensive program should solicit the help of an independent agency to assist in overseeing software development.
2. Focus Metrics on Managing the Program - Having metrics which focus on the purpose of managing the software development effort is critical to metrics effectiveness.
3. Implement Only the Most Useful Metrics - The program manager should only implement the most useful metrics which are absolutely required to manage the program.
4. Make the Software Developer Responsible for Metrics - To ensure that metrics are effective, they must be fully-integrated with the software development effort. One way that the program manager and contractor can

promote this integration is by ensuring that the software developer is also responsible for metrics.

5. Tailor Your Metrics (Management Level, Stage, and Presentation) - Metrics will be most effective if they are tailored to the specific application, such as management level, stage of development, and presentation.
6. Get Educated on Software Development and Metrics - When managing any software-intensive system it is vital that the program managers have at least a general understanding of software-related issues.
7. Cooperative Relationships Foster Success - When developing any system, having cooperative relationships with the contractor and other agencies involved greatly increases the probability of overcoming problems during development.

See Sections IV.D. for more detail.

6. How can these lessons learned be generalized to guide and support other program managers?

See answer to question 5 above or Section IV.D.

D. RECOMMENDATIONS FOR FURTHER STUDY

1. Software Metrics in Other Software-Intensive Systems

Research the use of software metrics in other software-intensive systems. Determine how metrics were use and the extent of their effectiveness. Develop a set of lessons learned based on an analysis of the specific case.

2. Program Manager Knowledge of Software Metrics

Conduct a survey of the knowledge base of program managers who manage software-intensive systems with respect to software related issues, including metrics. Analyze the results of the survey and any correlation to the program managers' self-assessed ability to manage their programs. Make recommendations for methods to improve or sustain the existing knowledge base.

3. Software Agencies

Investigate the variety of agencies available to assist program managers of software-intensive programs in managing their programs. Include an analysis of the extent of services that each agency can provide and their role in past or existing programs.

LIST OF REFERENCES

1. Albrecht, Allan J. and Gaffney, John E. Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, November 1983

2. Army Software Metrics Program Website, www.army.mil/swmetrics

3. Army Technology – The Website for Defence Industries – Army, www.army-technology.com/projects/bradley/

4. Boehm, Barry W., Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ, 1981

5. Booch, Grady, Object-Oriented Analysis and Design, Addison Wesley Longman, Inc., Menlo Park, CA, 1994

6. Booch, Grady, Software Engineering With Ada, Benjamin/Cummings Publishing Inc., Menlo Park, CA, 1987

7. Brooks, Frederick P. Jr., "The Mythical Man-Month", Datamation, December 1974

8. Bradley A3 Product Manager, Test and Evaluation Master Plan, Bradley Fighting Vehicle Systems Project Office, June 1995

9. Bradley A3 Product Manager Website, Bradley Fighting Vehicle Systems Project Office, www.pmbradley.com\A3\A3.htm

10. Conte, S. D., Dunsmore, H. E., and Shen, V. Y., Software Engineering Metrics and Models, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA, 1986

11. Cummings, Terry, MAJ (USA), Deputy Product Manager Bradley A3, Bradley Fighting Vehicle Systems Project Office, Personal Interview, 28 April 1998

12. Dalrymple, Edgar, Bradley Support Team, U.S. Army Aviation and Missile Command (AMCOM) Software Engineering Directorate, Personal Notes - Metrics Use on Bradley A3

13. Dalrymple, Edgar, Bradley Support Team, U.S. Army Aviation and Missile Command (AMCOM) Software Engineering Directorate, Telephone Interviews, 15 April 1998, 18 May 1998, and 29 May 1998

14. Defense Daily Network – The Network for Aerospace and Defense, www.defensedaily.com/progprof/army/bradley.html

15. DeMarco, Tom, Controlling Software Projects - Management, Measurement, & Estimation, Yourden Press, New York, NY, 1982

16. Department of Defense, Department of Defense Directive (DoDD) 5000.1 Defense Acquisition, Washington, D.C., GPO, 15 March 1996

17. Department of Defense, Department of Defense Regulation 5000.2-R Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information Systems (MAIS) Acquisition Programs, Washington, D.C., GPO, 15 March 1996 (Change 3, March 1998)

18. Department of Defense, Military Standard (MIL-STD-498) Software Development and Documentation, Washington, D.C., GPO, 5 December 1994

19. Fenton, Norman E. and Pfleeger, Shari L., Software Metrics: A Rigorous and Practical Approach, 2nd Edition, International Thomson Computer Press, London, UK, 1997

20. Frew, Frank, U.S. Army Aviation and Missile Command (AMCOM) Software Engineering Directorate, Telephone Interviews, 15 April 1998 and 18 May 1998

21. Grady, Robert B., Practical Software Metrics for Project Management and Process Improvement, PTR Prentice-Hall Inc., Englewood Cliffs, NJ, 1992

22. Johnson, Ted, LTC (USA), Product Manager Bradley A3, Bradley Fighting Vehicle Systems Project Office, Telephone Interview 21 April 1998, Personal Interview 28 April 1998

23. Jones, Capers, Patterns of Software System Failure and Success, International Thomson Computer Press, London, UK, 1996

24. Jones, Capers, Programming Productivity, McGraw Hill Inc., New York, NY, 1986

25. Liao, Chong K., Project Engineer Bradley A3, Bradley Fighting Vehicle Systems Project Office, Personal Interview, 28 April 1998

26. Parametric Cost Estimating Initiative Steering Committee, "Parametric Cost Estimating Handbook", Department of the Navy, Washington, D.C., GPO, Fall 1995

27. Perils, Alan, Sayward, Frederick, and Shaw, Mary, Software Metrics: An Analysis and Evaluation, MIT Press, Cambridge, MA, 1981

28. Reifer, Donald J., Software Management, IEEE Computer Society Press, Los Alamitos, CA, 1993

29. Rodgers, Kenneth P., "Embedded Software Development: A Case Analysis of the U.S. Army Bradley Fighting Vehicle A3 Program", Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1996

30. Software Engineering Institute, "Technical Report CMU/SEI-92-TR-19, Software Measurement for DoD Systems: Recommendations for Initial Core Measures", Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1992

31. Software Program Manager's Network, The Program Manager's Guide to Software Acquisition Best Practices, Version 2.0, Department of Defense, Washington D.C., GPO, April 1997

32. Software Technology Support Center, Guidelines for Successful Acquisition Management of Software-Intensive Systems, Version 2.0, Department of the Air Force, Hill AFB, Utah, June 1996

33. Stutzke, Richard D., "Software Estimating Technology: A Survey", Crosstalk, May 1996

34. Tacaks, Steve, ex-Chief, Systems Support, Ground Systems Division, United Defense, Limited Partnership, Telephone Interview, 21 May 1998

35. Tacaks, Steve, ex-Chief, Systems Support, Ground Systems Division, United Defense, Limited Partnership, Presentation, Naval Postgraduate School Systems Management Department, Monterey, CA, 14 August 98

36. Task Force Eagle and U.S. Army 1st Armored Division Website, www.tfeagle.army.mil/photos/Bradley.htm

37. Training and Doctrine Command (TRADOC) Systems Manager – Bradley Fighting Vehicle, "Bradley Program Overview, www.benning.army.mil/fbhome/tsm-b/bpo.htm, 20 May 1997

38. Training and Doctrine Command (TRADOC) Systems Manager – Bradley Fighting Vehicle, Fighting Vehicle Systems, Operational Requirements Document (ORD), M2/M3A3 Bradley, Ft. Benning, GA, 1993

39. United Defense, Limited Partnership, Ground Systems Division, Production Readiness Review Briefing, UDLP, Ground Systems Division, San Jose, CA, 3 March 1997

40. United Defense, Limited Partnership, Ground Systems Division, Software Development Plan (SDP), UDLP, Ground Systems Division, Santa Clara, CA, February 1996

41. United Defense, Limited Partnership, Ground Systems Division, Software Metrics Documentation, UDLP, Ground Systems Division, Santa Clara, CA, Jan 97 – Apr 98

42. United Defense, Limited Partnership, Ground Systems Division, Systems Engineering Management Plan (SEMP), UDLP, Ground Systems Division, Santa Clara, CA, December 1995

INITIAL DISTRIBUTION LIST

1. Defense Information Technical Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-5101

2. Dudley Knox Library 2
   411 Dyer Road
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Defense Logistics Studies Information Center 1
   U.S. Army Logistics Management Center
   Fort Lee, VA 23801-6043

4. OASA (RDA) 1
   AT7N: SARD-ZAC
   103 Army Pentagon
   Washington, DC 20310-0103

5. Prof David V. Lamm (Code SM/Lt) 4
   Naval Postgraduate School
   Monterey, CA 93943-5103

6. David F. Matthews, Col (Ret) (Code SM/Md) 2
   Naval Postgraduate School
   Monterey, CA 93943-5103

7. Prof Mark E. Nissen (Code SM/Ni) 1
   Naval Postgraduate School
   Monterey, CA 93943-5103

8. CPT James S. Romero 2
   3610 Hartsock Lane
   Colorado Springs, CO 80917